WTR/06-001

USAMRD/TR-2006-0001

# The Probitfit Program to Analyze Data from Laser Damage Threshold Studies

*Brian J. Lund*

Northrop Grumman Corporation
4241 Woodcock Drive, B-100
San Antonio, Texas 78228

United States Army Medical Detachment
Walter Reed Army Institute of Research
7965 Dave Erwin Drive
Brooks City-Base, Texas 78235



May 2006



Walter Reed Army Institute of Research
Silver Spring, Maryland 20910

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



**PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From - To)* |
|---|---|---|
| May 2006 | Final | Dec 2005 - May 2006 |

**4. TITLE AND SUBTITLE**

The ProbitFit Program to Analyze Data from Laser Damage Threshold Studies

**5a. CONTRACT NUMBER**

F41624-02-D-7003

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

62787

**6. AUTHOR(S)**

Lund, Brian J.

**5d. PROJECT NUMBER**

A878

**5e. TASK NUMBER**

878AA

**5f. WORK UNIT NUMBER**

N/A

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Northrop Grumman Corporation
4241 Woodcock Dr., Suite B-100
San Antonio, TX 78228

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

US Army Medical Research Detachment
Walter Reed Army Institute of Research
7965 Dave Erwin Drive
Brooks City-Base, TX 78235

**10. SPONSOR/MONITOR'S ACRONYM(S)**

USAMRD-WRAIR/MCMR-UWB

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

USAMRD/TR-2006-0001

**12. DISTRIBUTION/AVAILABILITY STATEMENT**

Unlimited Distribution

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**

The ProbitFit program has been developed at the US Army Medical Research Detachment to analyze dose-response data from laser damage threshold experiments. ProbitFit expands on the capabilities of the Probit program, developed at the USAF Armstrong Laboratory, while producing identical results when used to analyze the same data sets. ProbitFit implements the iterative probit analysis procedure developed by Finney. Probit analysis and the fitting procedure used by ProbitFit is described. The source code implementing the fitting routines as well as the procedures used to calculate the ED[P] values and associated fiducial limits are contained in an appendix.

**15. SUBJECT TERMS**

probit analysis, threshold, ED50

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Brian Lund |
| U | U | U | UU | 35 | 19b. TELEPHONE NUMBER *(Include area code)* (210) 536-4648 |

# Contents

# List of Figures



# List of Tables

## Disclaimer

The opinions or assertions contained herein are the private views of the author and are not to be construed as official or as reflecting the views of the Department of Defense, the Department of the Army, or Northrop Grumman. Citation of trade names in this report does not constitute an official endorsement or approval of the use of such items.

## Acknowledgments

The author would like to thank Gary Noojin for a number of useful discussions during the development of the ProbitFit program. Mr. Noojin also made the source code for the Probit program available to the author.

This work was supported by the U.S. Army Medical Research Detachment through Task Order 15 of U.S. Air Force Contract F41624-02-D-7003 (Northrop Grumman). This work was conducted at the U.S. Army Medical Research Detachment, Brooks City-Base, Texas.

# 1 Introduction

Probit analysis is the primary statistical tool used to analyze dose-response data from laser retinal damage threshold experiments. Beginning about 1970, the U.S. Army Medical Research Detachment (USARMD) performed this analysis using small programs developed in-house and based on the procedure described by Frisch [1]. Since 1996, USAMRD has been using computer programs that have evolved from the original EZ-probit program developed by the U.S. Air Force Armstrong Laboratory [2]. The current version of this program is Probit version 2.1.2.3. Although it has undergone minor corrections and revisions, Probit has existed essentially in its current form since 1998.

The author has developed a new program, ProbitFit, as a replacement for Probit. The primary enhancement offered by ProbitFit is an on-screen graph of the dose-response data, as well as the resulting fits. In addition, a rudimentary data editor is included, allowing the researcher the capability to rapidly explore several "what if" scenarios.

ProbitFit was written using the Java programming language, and therefore should run on any operating system for which an appropriate Java Runtime Environment is available. Although based on entirely new code, ProbitFit uses the same iterative fitting procedure developed by Finney [3] that is used by Probit. Therefore, as a primary design goal, ProbitFit was required to reproduce the results of Probit when used to process the same data set.

This report first provides a brief overview of probit analysis. The iterative routine used by ProbitFit to fit the dose-response data is then outlined, followed by a description of the fiducial limit calculations. There is no attempt at mathematical rigor; for details, the reader is referred to Finney [3].

# 2 Overview of Probit Analysis

**Probit Analysis** is a procedure used to analyze data in which the outcome of an experiment is quantal: there is a response (yes), or there is not a response (no). The *probability* of producing a response depends on the input parameters to the experiment. In an experiment to determine the laser-induced retinal damage threshold at a specified wavelength, pulse duration, and irradiance diameter, the input parameter is the energy of a pulse from the laser. A *yes* response corresponds to the observation of a lesion in the retina; *no* response means no alteration of the retina is observed.

## 2.1 Dose-Response Curve

In probit analysis, dose-response data is assumed to follow a **log-normal distribution**. That is, for a given dose $q$, the probability of a response is

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{x} \exp\left[\frac{-(t-\mu)^2}{2\sigma^2}\right] \mathrm{d}t \tag{1}$$

where $x = \log_{10}(q)$. The goal is to find the values of $\mu$ and $\sigma$ which best describe a set of experimental data.

Figure 1 shows the characteristic sigmoid shape of this distribution. The curve in this figure illustrates a normal distribution having a mean $\mu = 1.0$ and standard deviation $\sigma = 0.2$.

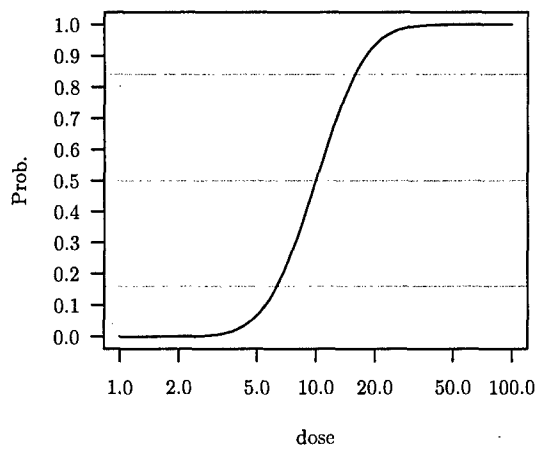


Figure 1: Example dose-response curve with a log-normal distribution having a mean $\mu = 1.0$ and a standard deviation $\sigma = 0.2$. These values correspond to $ED_{50} = 10^{\mu} = 10.0$ and $ED_{84}/ED_{50} = ED_{50}/ED_{16} = 10^{\sigma} = 1.58$. The gray lines indicate the 16% level $(\mu - \sigma)$, the 50% level $(\mu)$, and the 84% level $(\mu + \sigma)$.

The notation $\mathbf{ED_{50}}$ is used to refer to the the median effective dose, that is, dose at which the probability of producing a result is 50% (e.g. the laser pulse energy expected to produce a retinal lesion in half of the exposures). It is the $ED_{50}$ value obtained from a laser exposure experiment that is quoted as the damage threshold. From Equation 1, we can see that $P(x = \mu) = 0.5$. Therefore, the $ED_{50}$ is related to the mean of the probability distribution as $ED_{50} = 10^{\mu}$. The $ED_{50}$ extracted from the curve of Figure 1 is $ED_{50} = 10^{1} = 10.0$.

The steepness of the curve in Figure 1 is related to the standard deviation $\sigma$ of the distribution. For exposure data exhibiting a very sharp threshold, $\sigma$ will be very small, and the dose-response curve will approach a step function. If the exposure data is of poorer quality, $\sigma$ will be larger, and the dose response curve will have a finite slope.

For the log-dose value $x_{84} = \mu + \sigma$, the probability of producing a lesion is $P(x_{84}) = 0.84$. The $ED_{84}$ dose is given by $ED_{84} = 10^{\mu+\sigma}$. The ratio $ED_{84}/ED_{50} = 10^{\sigma}$ is directly related to the standard deviation of the dose-response distribution. In reports produced by USAMRD, the ratio $ED_{84}/ED_{50}$ is referred to as the "slope of the probit curve," and is used as a measure of how tightly a damage threshold has been determined from laser exposure data. Typically, $ED_{84}/ED_{50}$ is found to be in the range 1.0 to 2.0.

Note: At the log-dose value $x_{16} = \mu - \sigma$, the probability of producing a lesion is $P(x_{16}) = 0.16$. In this case $ED_{16} = 10^{\mu-\sigma}$, and therefore the ratio $ED_{50}/ED_{16} = 10^{\sigma}$ also gives a direct measure of the standard deviation of the dose-response distribution.

## 2.2 Probit Transformation

Making the change of variable

$$u = \frac{(t - \mu)}{\sigma} \tag{2}$$

in Equation 1 leads to the following expression for the probability distribution:

$$P(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{(x-\mu)}{\sigma}} \exp\left[\frac{-u^2}{2}\right] du \tag{3}$$

By using the **probit transformation**, defined as

$$Y - 5 = \frac{(x-\mu)}{\sigma} \tag{4}$$

the probability distribution (Equation 3) becomes

$$P(Y) = \frac{1}{2\pi} \int_{-\infty}^{Y-5} \exp\left[\frac{-u^2}{2}\right] du \tag{5}$$

The probit value $Y = 5$ corresponds to the mean log-dose $\mu$. Increasing the log-dose value $x$ by $1\sigma$ increases the probit value by 1.

> Note: The Normal Equivalent Deviate is defined as $Y = (x - \mu)/\sigma$, which takes on negative values for $x < \mu$. This made data analysis more difficult, especially before the wide availability of desktop computers. Defining the probit with an offset value of 5 generally limits the probit values to positive numbers. A probit of $Y = 0$ corresponds to $x = \mu - 5\sigma$. From Equation 1, $P(\mu - 5\sigma) = 2.9 \times 10^{-7}$, a probability value unlikely to be encountered in any realistic experiment.

Figure 2 shows the dose-response curve of Figure 1 replotted on a linear scale of probit values. The (now non-linear) percentage axis is drawn on the right side of the plot. The probit transformation (Equation 4) has linearized the data; the probit value is a linear function of the log-dose value:

$$Y = a + bx \tag{6}$$

The slope and intercept are (from Equation 4)

$$a = 5 - \mu/\sigma \tag{7}$$

$$b = 1/\sigma \tag{8}$$

Linearization makes it possible to extract information from the data using a graphical analysis [1]. After drawing a best-fit line by eye on a plot of probit vs. log-dose (Figure 2), $\mu$ and $\sigma$ are determined using Equations 7 and 8. However, the probit transformation also serves as the basis for the automated fitting procedure used by ProbitFit (Section 3).

> Note: The Probit program reports the value $b$ from Equation 6 as the "slope of the probit curve." From Section 2.1, $\sigma = \log_{10}(\mathrm{ED}_{84}/\mathrm{ED}_{50})$. Recall that the ratio $\mathrm{ED}_{84}/\mathrm{ED}_{50}$ is the USAMRD definition of the "slope" (Section 2.1). The two definitions of the "slope" are related through Equation 8:
>
> $$b = \frac{1}{\log_{10}(\mathrm{ED}_{84}/\mathrm{ED}_{50})} \tag{9}$$

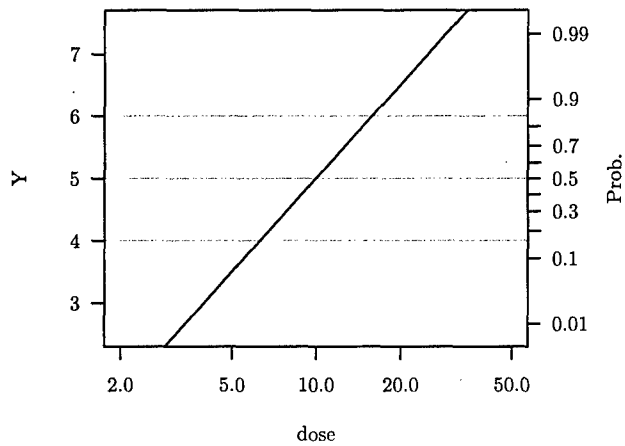


Figure 2: The dose-response curve of Figure 1 replotted on a linear scale of probit values (Equation 4). The right-hand axis shows the corresponding percentage values. The gray lines indicate the 16%, 50%, and 84% percentage levels.

## 2.3 Binomial Distribution and Exposures at a Single Dose

The log-normal distribution of Equation 1 gives the probability $P(x)$ of a lesion being produced by a single exposure. If $n$ exposures are made at the same dose, the probability of producing $r$ lesions is determined from the **binomial distribution**:

$$\Pr(r|n) = \frac{n!}{r!(n-r)!} P(x)^r \big(1 - P(x)\big)^{n-r} \tag{10}$$

The mean and variance in the expected number of lesions is

$$\bar{r} = nP(x) \tag{11}$$

$$\mathrm{Var}(r) = nP(x)\big(1 - P(x)\big) \tag{12}$$

## 2.4 Damage Threshold Experiments and Probit Analysis

An experimental determination of the damage threshold for laser-induced retinal injury generally consists of making exposures at a number of pulse energies in the retinas of one or more test subjects (usually non-human primates, or NHPs). The selected pulse energies are expected to bracket the damage threshold value. Multiple exposures at each pulse energy may be made. The retinal exposure sites are then examined ophthalmoscopically (usually 1 hour or 24 hours after exposure). The presence of a minimal visible lesion (MVL) is generally used to determine whether there was or was not a response to the particular laser pulse.

Assume $n_i$ exposures are made at the pulse energy (dose) $q_i$, and $r_i$ lesions are observed, for each of the $i = 1, \ldots, k$ pulse energies used in the experiment. From Equation 11, the ratio $p_i = r_i/n_i$ provides an estimate of the dose-response curve (Section 2.1) at the log-dose value $x_i = \log_{10}(q_i)$.

The experimentally determined dose-response curve, that is, the plot of the ratios $p_i$ versus the log-dose values $x_i$, is expected to exhibit the characteristic sigmoid shape of Figure 1. The goal of the probit analysis is to determine the values of the parameters $\mu$ and $\sigma$ such that Equation 1 best describes the data. The value for $ED_{50}$ and its fiducial limits (Section 4) may then be determined.

# 3 ProbitFit Fitting Procedure

ProbitFit uses an iterative fitting procedure to find the slope and intercept of the probit line. The mathematical derivation of the procedure, based on the Method of Maximum Likelihood, is described in detail by Finney [3]. It is difficult to work directly with the normal distribution curve of Equation 1 to derive the maximum likelihood equations. This is because, for example, partial derivatives of $P(x)$ with respect to $\mu$ and $\sigma$ are required. Therefore, the derivation of the fitting procedure is based on using the probit transformation (Equation 4) to linearize the data.

The total probability of observing $r_i$ lesions in $n_i$ trials at log-dose $x_i$ for each of the $i = 1, \ldots, k$ doses (pulse energies) is

$$\begin{aligned} \mathbb{P}(r_1, \ldots, r_k) &= \prod_{i=1}^{k} \Pr(r_i | n_i) \\ &= \prod_{i=1}^{k} \binom{n_i}{r_i} P(Y_i)_i^r \big(1 - P(Y_i)\big)^{n_i - r_i} \end{aligned} \tag{13}$$

It is assumed that the "true" dose-response distribution maximizes $\mathbb{P}$. The total probability $\mathbb{P}$ is proportional to $\exp(L)$, where the likelihood function is

$$L(a, b) = \sum_{i=1}^{k} r_i \log\big(P(Y_i)\big) + \sum_{i=1}^{k} (n_i - r_i) \log\big(1 - P(Y_i)\big) \tag{14}$$

The dependence of $L$ on the parameters $a$ and $b$ enters through Equations 5 and 6 for $P(Y)$. Maximizing the likelihood function $L$ will maximize the total probability $\mathbb{P}$. Doing this leads to an iterative procedure which, for each iteration, is identical to performing a weighted linear regression for the parameters $a$ and $b$.

## 3.1 Input

The input needed for the fitting procedure is the experimentally-determined dose-response data:

| | |
|---|---|
| $x_i$ | $\log_{10}$ of the $i^{th}$ dose level |
| $n_i$ | # of trials at dose level $x_i$ |
| $r_i$ | # lesions observed at dose level $x_i$ |
| $p_i = r_i/n_i$ | Experimental estimate for $P(x_i)$ |

Note, however, that the input to ProbitFit consists of the raw dose-lesion data, which lists, for each trial, the dose (beam or pulse energy) at which the trial was run, and whether a lesion was produced. The program will process the raw data to calculate the $x_i$, $n_i$, $r_i$ and $p_i$ values.

## 3.2 Iterative Fitting Procedure

The iterative fitting procedure used by ProbitFit can be outlined as follows:

1. Guess an initial value for the slope $b$ and intercept $a$. ProbitFit starts with $b = 0$ and $a = 5$.
2. Calculate $Y_i = a + bx_i$ for each log-dose value $x_i$.
3. Use Equation 5 to calculate $P_i = P(Y_i - 5)$ for each value $x_i$.
4. Calculate
$$z_i = \frac{1}{\sqrt{2\pi}} \exp\left[\frac{-(Y_i - 5)^2}{2}\right] \tag{15}$$
5. Calculate the **working probits**
$$y_i = Y_i + (p_i - P_i)/z_i \tag{16}$$
6. A weighted linear regression [4] of the working probits $y_i$ versus the log-doses $x_i$ is performed to find new, improved values for the slope $b$ and intercept $a$. The weight factor for each dose level is $n_i w_i$, where
$$w_i = \frac{z_i^2}{P_i(1 - P_i)} \tag{17}$$
7. Go back to step 2, repeat until convergence.

The mean $\mu = \log_{10}(\mathrm{ED}_{50})$ (Equation 1) is calculated at the end of each iteration. This is done by setting $Y = 5$ in Equation 6. Convergence is considered to occur when the change in $\mu$ from one iteration to the next is smaller than a user-controlled tolerance level.

> Note: ProbitFit actually produces a fit of the form $Y = \bar{y} + b(x - \bar{x})$, where $\bar{x}$ and $\bar{y}$ are weighted averages of the log-dose and working probit values. This is because $b$, $\bar{x}$ and $\bar{y}$ are used to calculate fiducial limits on $\mathrm{ED}_{50}$. (The weighting factors in calculating the averages are equal to $n_i w_i$, where $w_i$ is determined from Equation 17.)

## 3.3 Goodness of Fit, $\chi^2$

Although the fitting procedure produces a linear fit of the form of Equation 6, recall that the goal is to fit the experimental data to the log-normal distribution curve, Equation 1. Therefore, the measure of the goodness of the fit is

$$\chi^2 = \sum_{i=1}^{k} \frac{\big(p_i - P(x_i)\big)^2}{\mathrm{Var}(p_i)} \tag{18}$$

$P(x_i)$ is to be calculated using the final fit result. The weight of each point (dose level) is determined from the binomial distribution. Since $p_i = r_i/n_i$, then from Equation 12

$$\mathrm{Var}(p_i) = \frac{P(x_i)\big(1 - P(x_i)\big)}{n_i} \tag{19}$$

Therefore

$$\chi^2 = \sum_{i=1}^{k} n_i \frac{\left(p_i - P(x_i)\right)^2}{P(x_i)\left(1 - P(x_i)\right)} \tag{20}$$

Using equations 16 and 17, this can be reduced to a form useful for calculations:

$$\chi^2 = \sum_{i=1}^{k} n_i w_i \left(y_i - Y_i\right)^2 \tag{21}$$

where $y_i$ is the working probit, and $Y_i$ is calculated from the final fit. A statistical test of the quality of the fit is then provided by the $\chi^2$ distribution for $(k-2)$ degrees of freedom, where $k$ is the number of different pulse energies used in the experiment.

# 4 Fiducial Limits

The **fiducial limits** are an estimate of the range of values for the $ED_{50}$ which is supported by the statistical analysis of the data with some specified level of confidence.

## 4.1 Uncertainties in the Fit

The probit fitting procedure provides an equation to calculate the expected probit value for a given log-dose value $x$

$$Y = \bar{y} + b\left(x - \bar{x}\right) \tag{22}$$

The fitting procedure also provides estimates $\mathrm{Var}(\bar{y})$ and $\mathrm{Var}(b)$ for the variance in the fit parameters $\bar{y}$ and $b$. From this, the variance in the expected probit value may be determined:

$$\mathrm{Var}(Y) = \mathrm{Var}(\bar{y}) + \left(x - \bar{x}\right)^2 \mathrm{Var}(b) \tag{23}$$

Note that $\mathrm{Var}(Y)$ increases as the $x$ moves away from the mean value $\bar{x}$ of the log-dose values used in the exposure experiment.

## 4.2 Fiducial Limits for the Probit Value

In the usual statistical interpretation, experimentally determined probit values are expected to be normally distributed about a mean value $Y$ (Equation 22) with a standard deviation $\sigma = \sqrt{\mathrm{Var}(Y)}$. 68% of measured probit values are expected to lie within one $\sigma$ of $Y$. One says that the fiducial limits of $Y$ are given by $Y \pm \sqrt{\mathrm{Var}(Y)}$ at a confidence level of 68%.

More generally, the upper and lower fiducial limits to $Y$ may be calculated as

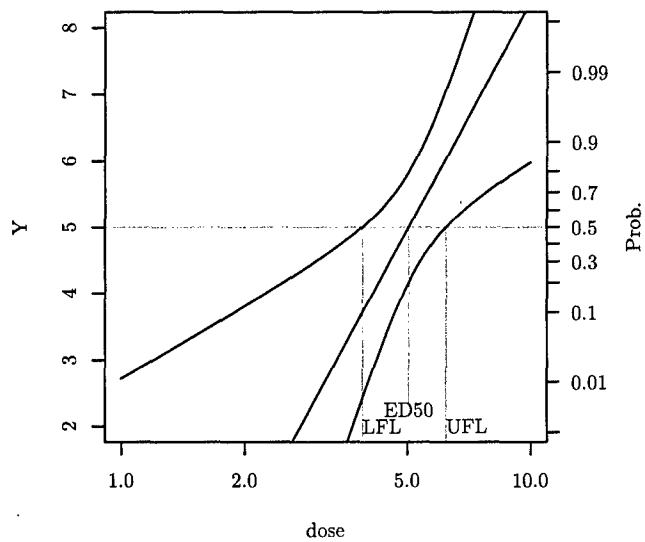


Figure 3: Example of a probit curve, including the 95% confidence level fiducial curves. The horizontal gray line indicates the 50% probability of response level. As indicated, the position at which this level intersects the probit curve and the two fiducial curves indicates the $ED_{50}$ and the Upper and Lower Fiducial Limits for the $ED_{50}$.

$$Y_{UFL} = Y + t\sqrt{\mathrm{Var}(Y)} \tag{24}$$

$$Y_{LFL} = Y - t\sqrt{\mathrm{Var}(Y)} \tag{25}$$

where $t$ is the normal deviate for the desired confidence level. That is, find $t$ such that

$$P_{norm}(t) = \frac{1}{2\pi}\int_{-\infty}^{t} \exp\left(-\frac{1}{2}u^2\right) \mathrm{d}u = 0.5 + \frac{\text{confidence level}}{2} \tag{26}$$

The value $t = 1.96$ corresponds to a 95% confidence level.

The far right side of Equation 26 can be understood as follows: $P_{norm}(1.96) = 0.975$, which means that the probability of finding a probit value $Y > Y_{UFL}$ is 2.5%. By the symmetry of the normal probability density function, the probability of finding a probit value $Y < Y_{LFL}$ is also 2.5%. Therefore, the probit value is expected to be in the range $Y_{LFL} \leq Y \leq Y_{UFL}$ with a probability of 95%.

Figure 3 shows the results of a probit analysis on a set of experimental laser exposure data. The 95% confidence level fiducial curves have been included. Note that they have the characteristic bowed shape expected from Equations 24, 25 and 23.

### 4.3 Fiducial Limits on the Dose

Of greater interest is the dose and the fiducial limits on that dose that produces a response with a specified probability level. Of particular interest here is the laser beam power or pulse energy that will produce a retinal lesion in 50% of exposures, i.e., the $ED_{50}$.

The method for locating the fiducial limits for the $ED_{50}$ is illustrated in Figure 3. The horizontal line in the figure indicates the 50% probability of response level. This level intersects the straight probit curve at the $ED_{50}$ dose level (as indicated by the dropped vertical line in the figure). The dose at which the 50% level intersects the fiducial curve to the left of the probit line is the Lower Fiducial Limit for the $ED_{50}$, while the dose at which this level intersects the fiducial curve to the right of the probit line gives the Upper Fiducial Limit for the $ED_{50}$.

Mathematically, Equations 24 and 25 are inverted to solve for the values $x_{UFL}$ and $x_{LFL}$ such that $Y_{UFL}$ and $Y_{LFL}$ are equal to the probit value corresponding to the desired response level. For the $ED_{50}$, this value is $Y = 5$. For the response probability $P$, the fiducial limits on $x_P = \log_{10}(ED_P)$ are

$$x_{UFL(LFL)} = x_P + \frac{g}{1-g}(x_P - \bar{x}) \pm \frac{t}{b(1-g)}\left[(1-g)\mathrm{Var}(\bar{y}) + (x_P - \bar{x})^2\mathrm{Var}(b)\right]^{\frac{1}{2}} \tag{27}$$

$$g = \frac{t^2}{b^2}\mathrm{Var}(b) \tag{28}$$

The +(-) corresponds to the upper (lower) limit. The upper and lower fiducial limits for the $ED_P$ are then $10^{x_{UFL}}$ and $10^{x_{LFL}}$.

Because of the shape of the fiducial curves (and also because the independent variable is the $\log_{10}$ of the dose), the fiducial limits will be asymmetric about $ED_P$. As Figure 3 indicates, the asymmetry increases the further the probability level is from 50%.

> If the *heterogeneity factor* $h = \chi^2/(k-2)$ is large (i.e. is significant as measured by the $\chi^2$ distribution for $k-2$ degrees of freedom, where $k$ is the number of exposure levels considered), then the variances in Equation 27 must be multiplied by $h$, and the factor $t$ is obtained from a Student's $t$-distribution. (See Finney [3] for details.)

## 5 Comparison of the ProbitFit and Probit Programs

In their report, Cain, Noojin and Manning [2] compared several procedures for performing a probit analysis with the Finney method (as implemented in Probit). Although they noted some differences in the results obtained by different methods, these differences were generally inconsequential compared to experimental uncertainties. In this section, we compare results from ProbitFit and Probit for three sample data sets. Numerical values are deliberately quoted to more decimal places than would normally be reported.

### 5.1 70 msec Pulse Data

This data set, taken from Table A-II of Frisch [1], is for 70 msec pulsed exposure data. The data has been binned by the pulse energy. Table 1 lists results of fitting this data set using ProbitFit and

Table 1: Comparison of fits to 70 msec pulse data

| Parameter | ProbitFit | Probit 2.1.2.3 |
|---|---|---|
| $ED_{50}$ | 12.400 | 12.400 |
| Upper FL | 13.751 | 13.751 |
| Lower FL | 10.848 | 10.848 |
| $ED_{84}$ | 18.568 | 18.568 |
| UpperFL | 22.280 | 22.280 |
| Lower FL | 16.536 | 16.536 |
| Slope of fit, $b$ | 5.67 | 5.67 |
| Iterations | 7 | 56 |

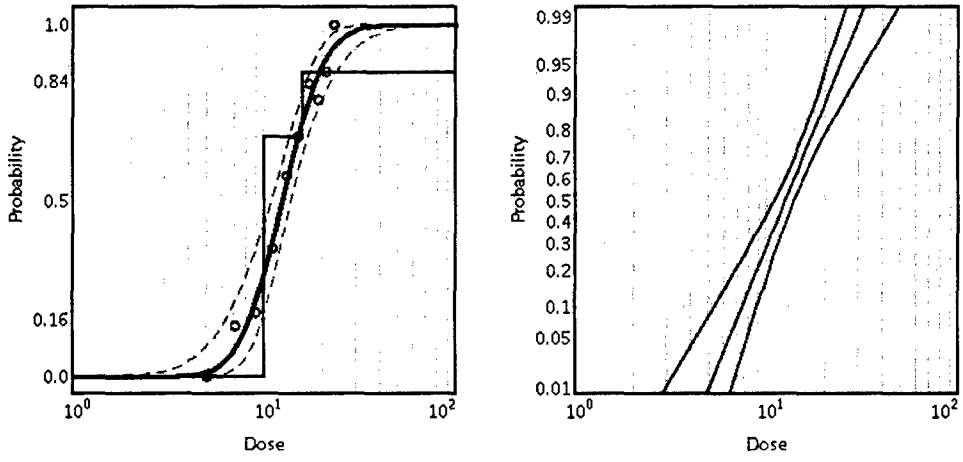


Figure 4: Dose-response and probability curve graphs of ProbitFit fits to 70 msec pulsed data [1].

Probit. Figure 4 shows a graph of the data set and the resulting fit which has been generated by the ProbitFit program.

The two programs have produced identical results, although the ProbitFit program required far fewer iterations. The programs were run with their default settings. In particular, the default tolerance used to test for convergence in the fitting procedure is far tighter in Probit (1 part in $10^{-18}$ for Probit versus 1 part in $10^{-9}$ for ProbitFit). The results obtained here indicate that the default tolerance used by Probit is far tighter than necessary to obtain an good fit to the data.

## 5.2 3.5 nsec Pulse Data at 540 nm

This data set was obtained for exposures at the wavelength 540 nsec for 3.5 nsec pulses [5]. In this case, the data was not pre-binned by pulse energy before analysis. A comparison of the results obtained using ProbitFit and Probit is presented in Table 2. Once again, the two program produce the same results. Graphs of the ProbitFit results are shown in Figure 5.

Table 2: Comparison of fits to 3.5 nsec pulse data at 540 nm

| Parameter | ProbitFit | Probit 2.1.2.3 |
|---|---|---|
| $ED_{50}$ | 6.218 | 6.218 |
| Upper FL | 7.208 | 7.208 |
| Lower FL | 5.503 | 5.503 |
| $ED_{84}$ | 7.990 | 7.990 |
| Upper FL | 11.083 | 11.083 |
| Slope of fit, $b$ | 9.13 | 9.13 |
| Iterations | 14 | 23 |

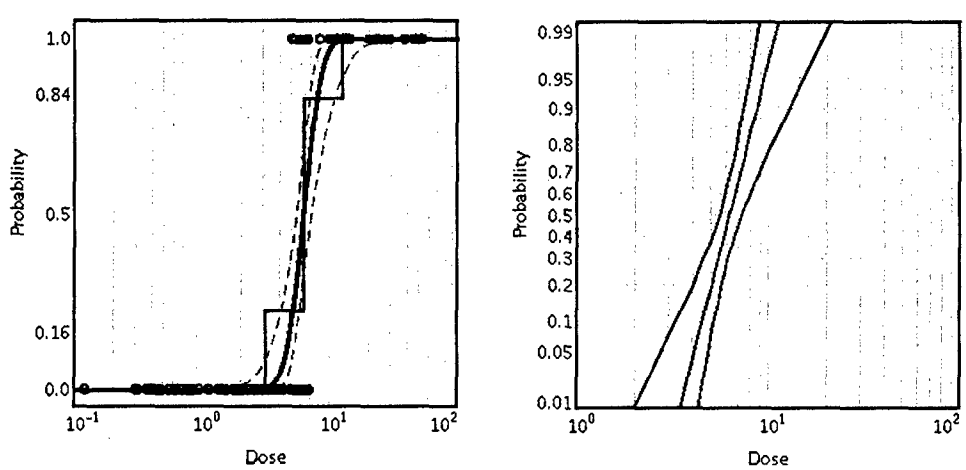


Figure 5: Dose-response and probability curve graphs of ProbitFit fits to 3.5 nsec pulsed data at $\lambda = 540$ nm [5].

Table 3: Comparison of fits 1.33 $\mu$m corneal threshold data

| Parameter | ProbitFit | Probit 2.1.2.3 |
|---|---|---|
| $ED_{50}$ | 233.188 | 233.188 |
| Upper FL | 78.987 | 78.982 |
| Lower FL | 327.742 | 327.744 |
| $ED_{84}$ | 506.640 | 506.640 |
| Upper FL | 1453.767 | 1432.849 |
| Slope of fit, $b$ | 2.95 | 2.95 |
| Iterations | 10 | 56 |

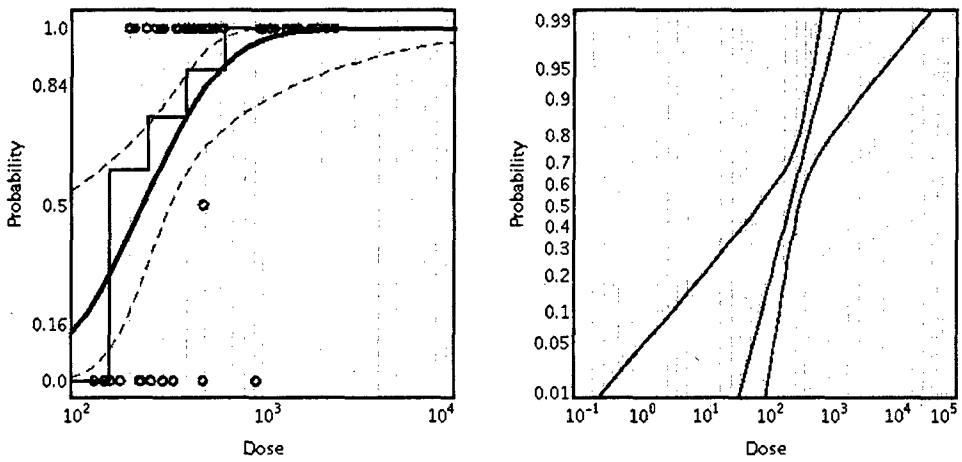


Figure 6: Dose-response and probability curve graphs of ProbitFit fits corneal threshold data at $\lambda = 1.33$ $\mu$m [6].

### 5.3 1.33 $\mu$m corneal threshold data

This data set was recorded for a measurement of the corneal damage threshold for exposures at a wavelength of 1.33 $\mu$m [6]. A comparison of the ProbitFit and Probit results is presented in Table 3. Figure 6 shows a graph of the ProbitFit results.

Although the $ED_{50}$ and $ED_{84}$ values are identical, there is a discrepancy at the $10^{-5}$ level between the fiducial limits calculated by the two programs. The origin of this discrepancy is not clear. It is unlikely to be related to the convergence test for the fit procedure, since the $EDp$ values are identical. The difference is considered insignificant for practical purposes.

## 6 Conclusion

The ProbitFit program to perform a probit analysis of dose-response data has been developed by the author for USAMRD. ProbitFit produces essentially identical results as Probit version 2.1.2.3 when used to analyze the same data set, while adding the significant capability to produce graphs of the data and resulting fits. ProbitFit can therefore be recommended as a new tool for the analysis of laser damage threshold experiments.

# References


[1] G.D. Frisch, "Quantal Response Analysis as Applied to Laser Damage Threshold Studies," Memorandum Report M70-27-1 of the Joint AMRDC-AMC Laser Safety Team, Dept. of the Army, Frankford Arsenal, Philadelphia, PA (1970).

[2] C.P. Cain, G.D. Noojin, L.M. Manning, "A Comparison of Various Probit Methods for Analyzing Yes/No Data on a Log Scale," USAF Technical Report AL/OE-TR-1996-0102. Brooks AFB, TX: USAF Armstrong Laboratory (1996).

[3] D.J. Finney, *Probit Analysis, 3rd. ed.*, Cambridge University Press, New York (1971).

[4] P.R. Bevington, *Data Reduction and Error Analysis for the Physical Sciences*, McGraw-Hill Book Company, New York (1969).

[5] D.J. Lund, P. Edsall, B.E. Stuck, "Wavelength Dependence of Laser-Induced Retinal Injury," SPIE Vol. 5688:383-389 (2005).

[6] J. Zuclich, P. Edsall, D. Lund, B.Stuck, "Ocular Effects and Safety Standard Implications for High-Power Lasers in 1.3-1.4 mm Wavelength Range," USAF Technical Report AFRL-HE-BR-TR-2004-0187. Brooks City-Base, TX: USAF Research Laboratory (2004).

[7] Sun Microsystems - Sun Developer Network web site (http://java.sun.com)

# A Selected Source Code Files

ProbitFit was written using the Java programming language. It was developed using the Java 1.5 JDK from Sun Microsystems [7]. Much of the source code deals with the user interface, and is not reproduced here. This appendix contains the source code for the classes used to store data, implement the iterative fitting procedure, and subsequently calculate the effective dose ($ED_p$) values and associated fiducial limits.

## A.1 TrialDataItem.java

```
package ng.probitfit.data;

/*
 * Class to hold the result of a single trial, i.e., the dose used,
 * and whether or not there a response was observed for that dose.
 */
public class TrialDataItem implements Comparable<TrialDataItem> {

    public double  dose;
    public boolean response;

    /*
     * Constructors
     */
     public TrialDataItem () {
        dose = 0.0;
        response = false;
    }

    public TrialDataItem (double d, boolean r) {
        dose = d;
        response = r;
    }

    /*
     * String representation for printing to file
     */
    public String toString () {
        return String.format ("%-10.3f %d", dose, response ? 1 : 0);
    }

    /*
     * Implementation of Comparable interface
     */
    public int compareTo (TrialDataItem tdi) {
        // order determined by 'dose' field
        return Double.compare (dose, tdi.dose);
    }
}
```

## A.2 TrialData.java

```
package ng.probitfit.data;

import java.util.*;

/*
 * A collection of TrialDataItem objects
 */
public class TrialData extends ArrayList<TrialDataItem> {

    // default data description
    private static String DEF_DESCRIPTION = "";

    // description of the data
    private String description = DEF_DESCRIPTION;

    /*
     * Clear the data
     */
    public void clear () {
        super.clear ();
        description = DEF_DESCRIPTION;
    }

    /* Set or get the description of the data */
    public void setDescription (String desc) {
        description = desc;
    }

    public String getDescription () {
        return description;
    }

    public void sort () {
        Collections.sort (this);
    }
}
```

## A.3 DoseResponseDataItem.java

```
package ng.probitfit.data;

/*
 * Class to hold a single probit data point of the dose-response curve.
 * Each data point includes the dose, number of trials at that dose,
 * number of responses at that dose, and the fraction of trials in
 * which a response is observed (= # responses / # trials)
 */
public class DoseResponseDataItem implements Comparable<DoseResponseDataItem> {

    public double dose;
```

```
    public int    trials;
    public int    responses;
    public double responseFraction;

    /*
     * Constructors
     */
    DoseResponseDataItem () {
        dose = 0.0;
        trials = 0;
        responses = 0;
        responseFraction = 0;
    }

    DoseResponseDataItem (double dose, int trials, int responses) {
        this.dose = dose;
        this.trials = trials;
        this.responses = responses;
        this.responseFraction = (double) trials / (double) responses;
    }

    DoseResponseDataItem (TrialDataItem tdi) {
        dose = tdi.dose;
        trials = 1;
        responses = (tdi.response) ? 1 : 0;
        responseFraction = (double) responses;
    }

    /*
     * String representation
     */
    public String toString () {
        return String.format ("%-10.2f%-7d%-4d",
                dose, trials, responses);
    }

    /*
     * Implementation of Comparable interface
     */
    public int compareTo (DoseResponseDataItem drdi) {
        // order by 'dose' field
        return Double.compare (dose, drdi.dose);
    }
}
```

## A.4 DoseResponseData.java

```
package ng.probitfit.data;

import java.util.*;

/*
```

```
 * Collection of DoseResponseDataItem objects
 */
public class DoseResponseData extends ArrayList<DoseResponseDataItem> {

    /*
     * Sort the items in the collection
     */
    public void sort () {
        Collections.sort (this);
    }

    /*
     * Factory method to create a DoseResponseData from
     * a TrialData object
     */
    public static DoseResponseData create (TrialData td) {
        // init return value
        DoseResponseData drd = new DoseResponseData ();

        // use a HashMap, keyed by the dose value, to organize the data
        HashMap<Double, DoseResponseDataItem> map =
            new HashMap<Double, DoseResponseDataItem> ();

        Iterator<TrialDataItem> iter = td.iterator ();
        while (iter.hasNext ()) {
            TrialDataItem tdi = iter.next ();
            Double key = new Double (tdi.dose);

            if (map.containsKey (key)) {
                // update # trials, # responses, and response fraction
                // at this dose
                DoseResponseDataItem drdi = map.get (key);
                drdi.trials++;
                if (tdi.response) drdi.responses++;
                drdi.responseFraction =
                    (double) drdi.responses / (double) drdi.trials;
            } else {
                // entry not found - add new item for this dose
                map.put (key, new DoseResponseDataItem (tdi));
            }
        }

        // insert contents of map into DoseResponseData and sort
        if (map.size () > 0) {
            Iterator<DoseResponseDataItem> mapIter =
                map.values ().iterator ();
            while (mapIter.hasNext ()) {
                drd.add (mapIter.next ());
            }

            drd.sort ();
        }

        return drd;
```

```
    }
}
```

## A.5 ProbitFit.java

```
package ng.probitfit.fit;

/*
 * Results of a Probit Fit
 * Includes slope and intercept of the fit line,
 * variance of slope and intercept,
 * ybar (weighted avg. of working probit values) and its variance,
 * xbar (avg. of log10(dose)),
 * chi-squared of fit,
 * # degrees of freedom
 *
 * All these values are neede to calculate ED(prob) values, and the
 * associated fiducial limits.
 */
public class ProbitFit {

    public double a;     // intercept
    public double b;     // slope
    public double varA;  // variance of intercept
    public double varB;  // variance of slope
    public double varYbar; // variance in weighted mean of probits
    public double chi2;    // chi-squared of fit
    public int    degFreedom;  // degrees of freedom (= # pts - 2)

    public double xbar;  // weighted avg of log10(dose) (for fid. limits)
    public double ybar;  // weighted avg of probit values (diagnostic)
    public int    numIterations;  // # iterations required for fit

    public double Sxy;   // diagnostic, used to calc. fit parameters
    public double Syy;   //    (varB = 1/Sxx, varYBar = 1/S0)

    public boolean infiniteSlope;  // set to 'true' if the slope b meets
                                   // some program-defined criterion
                                   // to be considered 'inifinite'
                                   // This affects fid. limit calcs.

    /*
     * Default constructor
     */
    public ProbitFit () {
        a = 0.0;
        b = 0.0;
        varA = 0.0;
        varB = 0.0;
        varYbar = 0.0;
        chi2 = 0.0;
        degFreedom = 0;
```

```
        xbar = 0.0;
        ybar = 0.0;
        numIterations = 0;
        Sxy = 0.0;
        Syy = 0.0;
        infiniteSlope = false;
    }

    /*
     * Copy constructor
     */
    public ProbitFit (ProbitFit pf) {
        a = pf.a;
        b = pf.b;
        varA = pf.varA;
        varB = pf.varB;
        varYbar = pf.varYbar;
        chi2 = pf.chi2;
        degFreedom = pf.degFreedom;
        xbar = pf.xbar;
        ybar = pf.ybar;
        numIterations = pf.numIterations;
        Sxy = pf.Sxy;
        Syy = pf.Syy;
        infiniteSlope = pf.infiniteSlope;
    }

    /*
     * String representation, for debugging purposes
     */
    public String toString () {
        StringBuffer sb = new StringBuffer ("a=");
        sb.append (a); sb.append ("    ");
        sb.append ("b="); sb.append (b); sb.append ("\n");
        sb.append ("varA="); sb.append (varA); sb.append ("    ");
        sb.append ("varB="); sb.append (varB); sb.append ("\n");
        sb.append ("chi2="); sb.append (chi2); sb.append ("    ");
        sb.append ("degFreedom="); sb.append (degFreedom);

        return sb.toString ();
    }
}
```

## A.6 ProbitFitProcedure.java

```
package ng.probitfit.fit;

import java.util.*;

import ng.probitfit.data.*;

/*
```

```
 * Class that performs the probit fitting procedured
 */
public class ProbitFitProcedure {

    // default values for fit procedure parameters
    private static double DEF_TOLERANCE = 1.0e-9;
    private static int    DEF_MAXITER   = 55;

    // Slopes greater than this value should be flagged as 'infinite'
    private static double INFINITESLOPE = 200.0;

    // class fields
    private WorkingData wd;
    private double         tol = DEF_TOLERANCE;
    private int            maxIter = DEF_MAXITER;

    private ProbitFit probitFit = new ProbitFit ();

    /*
     * Constructor - requires a DoseResponseDataSet as input
     */
    public ProbitFitProcedure (DoseResponseData drd) {
        wd = new WorkingData (drd);

        probitFit.degFreedom = wd.size () - 2;
    }

    /*
     * Accessors to get/set fit procedure parameters
     */
    public void   setTolerance (double tolerance) { tol = tolerance; }
    public double getTolerance ()                 { return tol; }

    public void setMaxIterations (int n) { maxIter = n; }
    public int  getMaxIterations ()      { return maxIter; }

    /*
     * Return results of fit
     */
    public ProbitFit getFit () { return probitFit; }

    /*
     * Perform the fit procedure
     * Returns # iterations if fit converges
     *         -1 if max. iterations reached without convergence
     */
    public int doFit () {
        // initial guess for fit equations is intercept a = 5.0,
        // slope b = 0.0, i.e., all Yp = 5
        probitFit.a = 5.0;
        probitFit.b = 0.0;

        // Convergence of fit process is determined by considering the
        // change in calculated log10(ED50) values between iteration steps
```

```
        double x50 = 0.0;
        double oldX50;

        probitFit.numIterations = 0;
        do {
            oldX50 = x50;
            // update weights, working probits, etc for current fit
            wd.updateForFit (probitFit.a, probitFit.b);
            iterationStep ();
            x50 = (5.0 - probitFit.a) / probitFit.b;
            if (probitFit.b > INFINITESLOPE) probitFit.infiniteSlope = true;

            ++probitFit.numIterations;
            if (probitFit.numIterations >= maxIter) {
                // flag no convergence in prescribe # of iterations ...
                probitFit.numIterations = -1;
                break;      // ... and break out of loop
            }
        } while ((Math.abs (x50 - oldX50) > tol) && !probitFit.infiniteSlope);

        return probitFit.numIterations;
    }

    /*
     * Perform one step of the iterative fitting procedure
     */
    private void iterationStep () {
        double Sx = 0.0;  // sum of w*x
        double Sy = 0.0;  // sum of w*ywp
        double S  = 0.0;  // sum of w
        // iterate through data to find xbar (avg of log10(dose)) and
        // ybar (avg of working probit values)
        Iterator<WorkingDataItem> iter = wd.iterator ();
        while (iter.hasNext ()) {
            WorkingDataItem wdi = iter.next ();
            S += wdi.w;
            Sx += wdi.w * wdi.x;
            Sy += wdi.w * wdi.ywp;
        }

        double xbar = Sx / S;
        double ybar = Sy / S;

        // now iterate to calculate Sxx = sum (w * (x-xbar)^2),
        // Sxy = sum (w*(x-xbar)*(y-ybar)), Syy = sum (w*(y-ybar)^2)
        double Sxx = 0.0;
        double Sxy = 0.0;
        double Syy = 0.0;
        double Sb  = 0.0;  // sum (w*(x-xbar)*ywp)

        iter = wd.iterator ();  // reset iterator to beginning of data
        while (iter.hasNext ()) {
            WorkingDataItem wdi = iter.next ();
            double tx = (wdi.x - xbar);
```

```
            double ty = (wdi.ywp - ybar);
            Sxx += wdi.w * tx * tx;
            Sxy += wdi.w * tx * ty;
            Syy += wdi.w * ty * ty;
            Sb  += wdi.w * tx * wdi.ywp;
        }

        // now calculate intercept, slope, variances, and chi^2
        probitFit.b       = Sb / Sxx;
        probitFit.a       = ybar - probitFit.b * xbar;
        probitFit.varB    = 1.0 / Sxx;
        probitFit.varYbar = 1.0 / S;
        probitFit.varA    = 1.0 / S + xbar * xbar / Sxx;
        probitFit.chi2    = Syy - Sxy * Sxy / Sxx;
        if (probitFit.chi2 < 0.0)
            probitFit.chi2 = 0.0;  // pathological data sets
        probitFit.xbar    = xbar;
        probitFit.ybar    = ybar;
        probitFit.Sxy     = Sxy;
        probitFit.Syy     = Syy;
    }
}
```

## A.7 WorkingDataItem.java

```
package ng.probitfit.fit;

import ng.math.*;
import ng.probitfit.data.*;

/*
 * Hold information from a single trial for the probit fit procedure.
 * Holds log10(dose), # trials, and observed probability of response.
 * Also holds the probit value from the current fit iteration
 * (Y = a + b*x), P(Y), z, weight, and working probit value
 */
class WorkingDataItem implements Comparable<WorkingDataItem> {

    public double x;    // log10(dose)
    public int    n;    // # trials at dose
    public double Pobs; // observed # responses / # trials at dose

    public double Yp;   // probit value for current fit iteration
    public double z;    // prob. density for Yp (from normal density fn)
    public double Py;   // prob. for Yp (from normal distribution)
    public double w;    // weight for this point
    public double ywp;  // working probit value

    private static final double MINP = 10.0 * Double.MIN_VALUE;

    /* constructors */
    public WorkingDataItem (double dose, int n, double prob) {
```

```
        x = Functions.log10 (dose);
        this.n = n;
        Pobs = prob;
        updateForFit (5.0, 0.0);
    }

    public WorkingDataItem (DoseResponseDataItem drdi) {
        x  = Functions.log10 (drdi.dose);
        n  = drdi.trials;
        Pobs = drdi.responseFraction;
        updateForFit (5.0, 0.0);
    }

    /*
     * Calculate probit values, weight, etc. for a specified
     * intercept a and slope b
     */
    public void updateForFit (double a, double b) {
        // Probit value based on current intercept and slope
        Yp = a + b * x;

        double ypm5 = Yp - 5.0;
        z = Distributions.dnorm (ypm5);
        Py = Distributions.pnorm (ypm5);

        if (Math.abs (Pobs - Py) <= MINP) {
            w = 0.0;
            ywp = Yp;
        } else {
            w = (double) n * z * z / (Py * (1.0-Py));
            ywp = Yp + (Pobs - Py) / z;
        }
    }

    /*
     * Comparable interface method
     */
    public int compareTo (WorkingDataItem wdi) {
        // order by x {log10(dose)}
        return Double.compare (x, wdi.x);
    }
}
```

## A.8 WorkingData.java

```
package ng.probitfit.fit;

import java.util.*;

import ng.probitfit.data.*;

/*
```

```
 * Collection of WorkingData, used to perform the Probit Fit
 */
class WorkingData extends ArrayList<WorkingDataItem> {

    /*
     * Constructor - creates a WorkingData object from a
     *               DoseResponseData
     */
    public WorkingData (DoseResponseData drd) {
        Iterator<DoseResponseDataItem> iter = drd.iterator ();
        while (iter.hasNext ()) {
            add (new WorkingDataItem (iter.next ()));
        }
    }

    /*
     * Sort the collection
     */
    public void sort () {
        Collections.sort (this);
    }

    /*
     * Update the WorkingDataItem items in this collection for the
     * current iterations's fit parameters
     */
    public void updateForFit (double a, double b) {
        Iterator<WorkingDataItem> iter = iterator ();
        while (iter.hasNext ()) {
            iter.next ().updateForFit (a, b);
        }
    }
}
```

## A.9 FitProcedureOptions.java

```
package ng.probitfit.fit;

/*
 * Class to hold options controlling the fit procedure
 */
public class FitProcedureOptions {

    // default settings
    private static final int DEF_TOLEXPONENT   = 9;
    private static final int DEF_MAXITERATIONS = 55;

    // data fields
    public int tolExponent;  // fit procedure convergence test uses
                             // tol = 10^(-tolExponent)
    public int maxIterations;
```

```
    /*
     * Constructors
     */
    public FitProcedureOptions () {
        tolExponent   = DEF_TOLEXPONENT;
        maxIterations = DEF_MAXITERATIONS;
    }

    public FitProcedureOptions (FitProcedureOptions fpo) {
        tolExponent   = fpo.tolExponent;
        maxIterations = fpo.maxIterations;
    }

}
```

## A.10 EDEvaluator.java

```
package ng.probitfit.fit;

import ng.math.*;

/*
 * Class to calculate ED(prob) values using the results of a Probit Fit.
 */
public class EDEvaluator {

    // data fields
    private ProbitFit pf;              // result of fit
    private double    confLevel;       // fiducial limit confidence level
    private double    heteroTestLevel; // test for heterogeneity

    private double h;  // heterogeneity factor
    private double t;  // value of normal or student distribution
                       // corresponding to fiducial confidence level

    private double g;      // factor for fiducial limit calculations
    private double onemg;  // (1.0 - g)
    private double gd1mg;  // g/(g-1), factor in fid. limit calculation

    private double facSqrt; // (t*sqrt(h))/(b*(1-g)), in fid. limit calc.

    private boolean initialized;  // flag for lazy eval of params

    /*
     * Constructor
     */
    public EDEvaluator (ProbitFit probFit, FiducialLimitOptions options) {
        pf = probFit;

        confLevel = options.confidenceLevel;
        heteroTestLevel = options.heterogeneityTest;
```

```
    // indicate that h, t, g, gm1 need to be calculated
    initialized = false;
}

/*
 * Read-only accessor for intermediate calculation factors
 */
public double getH () {
    if (!initialized)
        init ();
    return h;
}

public double getG () {
    if (!initialized)
        init ();
    return g;
}

public double getT () {
    if (!initialized)
        init ();
    return t;
}

/*
 * Return ED and fiducial limits for requested probablity level
 */
public ED getED (double probability) {
    if (!initialized)
        init ();

    // get probit corresponding to probability, recalling that
    // the probit value is defined as the value y such that
    // pnorm (y-5) = probability
    double y = Distributions.qnorm (probability) + 5.0;

    // log10(ED)
    double x = (y -pf.a) / pf.b;

    // log10(fiducial limits)
    double xUpper = x;
    double xLower = x;

    if (!pf.infiniteSlope) {
        // terms for fiducial limits
        double xmxbar = x - pf.xbar;

        double fA;
        double fB = onemg * pf.varYbar + xmxbar * xmxbar * pf.varB;
        if (fB <= 0.0) {
            // safety check - not sure if this is possible
            // probably need some really screwed up data
            fB = 0.0;
```

```
                fA = 0.0;
            } else {
                fB = facSqrt * Math.sqrt (fB);
                fA = gd1mg * xmxbar;  // g * (x-xbar) / (g-1)
            }

            // log10(fiducial limits)
            xUpper += (fA + fB);
            xLower += (fA - fB);
        }

        ED ed = new ED ();
        ed.ed      = Functions.tenTo (x);
        ed.upperFL = Functions.tenTo (xUpper);
        ed.lowerFL = Functions.tenTo (xLower);

        return ed;
    }

    /*
     * Calculate t and heterogeneity factor h
     */
    protected void init () {
        double nu = (double) pf.degFreedom;
        double px2 = Distributions.pchi2 (pf.chi2, nu);  // chi^2 prob.

        double probLevel = 0.5 + confLevel/2.0;
        if (px2 < heteroTestLevel) {
            // use heterogeneity factor, student distribution
            h = pf.chi2 / nu;
            t = Distributions.qstudent (confLevel, nu);
        } else {
            // use normal distribution
            h = 1.0;
            t = Distributions.qnorm (probLevel);
        }

        double temp = t / pf.b;  // h * t^2 / b^2

        g = h * temp * temp * pf.varB;

        onemg = 1.0 - g;
        gd1mg = g / onemg;

        facSqrt = Math.sqrt (h) * temp / onemg;

        initialized = true;
    }
}
```

## A.11 ED.java

```
package ng.probitfit.fit;

/*
 * ED(prob) level and associated fiducial limits
 */
public class ED {
    public double ed;
    public double upperFL;  // upper fiducial limit
    public double lowerFL;  // lower fiducial limit
}
```

## A.12 FiducialLimitOptions.java

```
package ng.probitfit.fit;

/*
 * Parameters used when calculating fiducial limits
 */
public class FiducialLimitOptions {

    // default values of the parameters
    private static final double DEF_CONF_LEVEL  = 0.95;
    private static final double DEF_HETERO_TEST = 0.10;

    // data fields
    public double confidenceLevel;
    public double heterogeneityTest;

    /*
     * Default constructor
     */
    public FiducialLimitOptions () {
        confidenceLevel   = DEF_CONF_LEVEL;
        heterogeneityTest = DEF_HETERO_TEST;
    }

    /*
     * Copy constructor
     */
    public FiducialLimitOptions (FiducialLimitOptions flo) {
        confidenceLevel   = flo.confidenceLevel;
        heterogeneityTest = flo.heterogeneityTest;
    }
}
```

## A.13 Distributions.java

```
package ng.math;

/*
 * Class to return probability distributions, densities, and quantiles
 */
public class Distributions {

    // Useful constants
    private static final double SQRTOF2 = Math.sqrt(2.0);
    private static final double SQRTOF2PI = Math.sqrt(2.0 * Math.PI);
    private static final double SQRTPI = Math.sqrt(Math.PI);

    // Class data
    private Functions func = new Functions();
    private static double tol = 1.0e-8;

    /*
     * Set or get tolerance for numerical routines
     */
    public static void setTolerance(double tolerance) { tol = tolerance; }
    public static double getTolerance() { return tol; }

    /*
     * Normal distributions
     *    pnorm(x) = normal prob. distribution
     *    dnorm(x) = normal prob. density function
     *    qnorm(p) = quantile for normal prob. dist.
     */
    public static double pnorm(double x) {
        return (1.0 + Functions.erf(x/SQRTOF2)) / 2.0;
    }

    public static double dnorm(double x) {
        return Math.exp(-x*x/2.0) / SQRTOF2PI;
    }

    public static double qnorm(double p) {
        // Use Newton-Raphson method to find the value x for which
        // pnorm(x) = p
        // Testing indicates # iterations < 10 for values of interest
        // in the probit program
        double x = 0.0; // initial guess
        double dx;
        do {
            dx = (pnorm(x) - p) / dnorm(x);
            x -= dx;
        } while (Math.abs(dx) > tol);

        return x;
    }
```

```
    /*
     * Chi-square distribution
     *   Probability of observing a larger value of chi-squared than the
     *   input parameter chi2 for the nu degrees of freedom
     *   pchi2 large -> prob of observing a larger chi-squared is high
     *               -> good fit
     */
    public static double pchi2(double chi2, double nu) {
        return 1.0 - Functions.gammap(nu/2.0, chi2/2.0);
    }

    /*
     * Student's distribution
     *    pstudent - prob. function
     *    qstudent - quantile
     */
    public static double pstudent(double t, double nu) {
        double x = nu / (nu + t*t);
        return 1.0 - Functions.betainc(nu/2.0, 0.5, x);
    }

    public static double qstudent(double p, double nu) {
        // pre-calculate some common factors for the derivative
        double nup1d2 = (nu + 1.0)/2.0;
        double nud2 = nu/2.0;
        double fac1 = Math.log(2.0/SQRTPI)
            + Functions.gammaln(nup1d2)
            - Functions.gammaln(nud2)
            + nud2*Math.log(nu);

        double q = 0.0; // initial guess
        double dq;
        do {
            double fac2 = fac1 - nup1d2*Math.log(nu + q*q);
            dq = (pstudent(q,nu) - p)/Math.exp(fac2);
            q -= dq;
        } while (Math.abs(dq) > tol);

        return q;
    }

}
```

## A.14 Functions.java

```
package ng.math;

/*
 * Class provides special functions
 *
 * Includes:
 *   erf(x)         Error function
```

```
 *   log10(x)       Log based 10
 *   tenTo(x)       10^x
 *   gammaln(x)     log of Gamma function
 *   gammap(a,x)    incomplete gamma function
 *   betainc(a,b,x) incomplete beta function
 */
public class Functions {

    // Useful constants
    private static final double SQRTPI = Math.sqrt(Math.PI);
    private static final double SQRT2PI = Math.sqrt(2.0*Math.PI);
    private static final double LOGOF10 = Math.log(10.0);
    private static final double TINY = 1.0e-30;

    // Class data
    private static double tol = 1.0e-14;  // tolerance for numerical routines

    /*
     * Set or get tolerance for numerical calculations
     */
    public static void setTolerance(double tolerance) { tol = tolerance; }
    public static double getTolerance() { return tol; }

    /*
     * erf(x) - Error function
     */
    public static double erf(double x) {
        double sign = 1.0;
        if (x < 0.0) {
            x = Math.abs(x);
            sign = -1.0;
        }

        // Testing indicates series solution requires fewer iteration for
        // x < 2.5, while continued fraction converges faster for x > 2.5
        if (x < 2.5) {
            return sign * erfSeries(x);
        } else {
            return sign * (1.0 - erfcContFrac(x));
        }
    }

    // Series solution for error func - called by erf(x)
    private static double erfSeries(double x) {
        // Series expansion from Abramowitz and Stegun
        int k = 0;
        double an = 1.0; // series term coeff.
        double sum = 1.0; // value from 1st term of series
        double oldsum;

        // Not checking # iterations here - the erf(x) routine only calls this
        // method for values of x < 2.5, a region for which at max 30-40
        // iterations are required
        do {
```

```
            oldsum = sum;
            ++k;
            an *= -x*x;
            an /= (double) k;
            sum += an / (2.0 * k +1.0);
        } while (Math.abs(sum - oldsum) > tol);

        return 2.0 * x * sum / SQRTPI;
    }

    // Continued fraction solution for complimentary error function -
    // called by erf(x)
    private static double erfcContFrac(double x) {
        // CF expression from Abramowitz and Stegun
        // Lentz's method to evalute CF base on material from
        // Numerical recipes in C
        double c = x + 1.0/TINY; //start from b0 = 0
        double d = 1.0/x;
        double delta = c * d;
        double f = TINY * delta; // This takes care of 1st iteration

        int i = 1;
        double a = 1.0; // all bn = x for n > 0
        do {
            ++i;
            a = (double) (i-1) / 2.0;
            // recursion relations for cn and dn
            d = x + a*d;
            if (d == 0.0) d = TINY; // Dangerous check
            d = 1.0/d;
            c = x + a/c;
            if (c == 0.0) c = TINY; // Dangerous check
            delta = c * d;
            f = f * delta;
        } while (Math.abs(delta-1.0) > tol);

        return f * Math.exp(-x*x)/SQRTPI;
    }

    /*
     * log10(x) - log based 10 of x
     * tenTo(x) - 10^x
     */
    public static double log10(double x) {
        return Math.log(x) / LOGOF10;
    }

    public static double tenTo(double x) {
        return Math.exp(x * LOGOF10);
    }

    /*
     * Natural log of gamma functions
     */
```

```
private static final double gammaln_coeff[] = {
    76.18009172947146,
    -86.50532032941677,
    24.01409824083091,
    -1.231739572450155,
    0.1208650973866179e-2,
    -0.5395239384953e-5
};
public static double gammaln(double x) {
    // Routine adapted from Numerical Recipes in C
    double temp = x + 5.5;
    temp -= (x+0.5)*Math.log(temp);
    double y = x;
    double sum = 1.000000000190015;
    for (int i = 0; i < 6; i++) {
        ++y;
        sum += gammaln_coeff[i]/y;
    }
    return -temp + Math.log(SQRT2PI * sum / x);
}

/*
 * Incomplete gamma functions P(a,x)
 */
public static double gammap(double a, double x) {
    // Routine adapted from Numerical Recipes in C
    if (x < 0.0) {
        throw new IllegalArgumentException("gammap: Illegal arg, x=" + x);
    }
    if (a < 0.0) {
        throw new IllegalArgumentException("gammap: Illegal arg, a=" + a);
    }

    if (x < a + 1.0) {
        return gamSeries(a,x);
    } else {
        return 1.0 - gamContFrac(a,x);
    }
}

// Support functions for gammap
private static double gamSeriesTolerance = 1.0e-10;
private static double gamSeries(double a, double x) {
    // Series solution for P(a,x)
    double t = 1.0/a;
    double sum = t;
    double oldsum;
    double aa = a;
    do {
        oldsum = sum;
        ++aa;
        t *= (x/aa);
        sum += t;
    } while (Math.abs(sum - oldsum) > gamSeriesTolerance);
```

```
    return sum * Math.exp(-x + a*Math.log(x) - gammaln(a));
}

private static int GAMCF_MAXITER = 100;
private static double gamContFrac(double a, double x) {
    // Constant fraction for 1.0 - P(a,x)
    // Routine adapted from Numerical Recipes in C
    double b = x + 1.0 - a;
    double c = 1.0/TINY;
    double d = 1.0/b;
    double h = d;
    double delta;

    int nIter = 0;
    do {
        ++nIter;
        double an = -nIter * (nIter - a);
        b += 2.0;
        d = an*d + b;
        if (Math.abs(d) < TINY) d = TINY;
        c = b + an/c;
        if (Math.abs(c) < TINY) c = TINY;
        d = 1.0/d;
        delta = d*c;
        h *= delta;
        if (nIter >= GAMCF_MAXITER) {
            throw new RuntimeException("gamConstFrac: No convergence");
        }
    } while (Math.abs(delta - 1.0) > tol);

    return Math.exp(-x + a*Math.log(x) - gammaln(a))*h;
}

/*
 * Incomplete beta function Ix(a,b)
 */
public static double betainc(double a, double b, double x) {
    // Routine from Numerical Recipes in C
    if (x < 0.0 || x > 1.0) {
        throw new IllegalArgumentException("betainc: Illegal arg x=" + x);
    }

    double bt;
    if (x == 0.0 || x == 1.0) {
        bt = 0.0;
    } else {
        bt = gammaln(a+b)-gammaln(a)-gammaln(b)
            + a*Math.log(x)+b*Math.log(1.0-x);
        bt = Math.exp(bt);
    }

    if (x < (a+1.0)/(a+b+2.0)) {
        // Use continued fraction directly
```

```
            return bt * betaContFrac(a,b,x)/a;
        } else {
            // Use continued fraction after making symmetry transformation
            return 1.0 - bt * betaContFrac(b,a,1.0-x)/b;
        }
    }

    // Continued fraction for incomplete beta
    private static int BETACF_MAXITER = 100;
    private static double betaContFrac(double a, double b, double x) {
        // Routine from Numerical Recipes in C
        double qab = a + b;
        double qap = a + 1.0;
        double qam = a - 1.0;

        double c = 1.0;
        double d = 1.0 - qab*x/qap;
        if (Math.abs(d) < TINY) d = TINY;
        d = 1.0/d;
        double h = d;

        int nIter = 0;
        double delta;
        do {
            ++nIter;
            double m2 = 2.0 * nIter;
            double aa = nIter*(b-nIter)*x/((qam+m2)*(a+m2));
            // One step (even one) of the recurrence
            d = 1.0 + aa*d;
            if (Math.abs(d) < TINY) d = TINY;
            d = 1.0/d;
            c = 1.0 + aa/c;
            if (Math.abs(c) < TINY) c = TINY;
            h *= d*c;

            // One step (odd one) of the recurrence
            aa = -(a+nIter)*(qab+nIter)*x/((a+m2)*(qap+m2));
            d = 1.0 + aa*d;
            if (Math.abs(d) < TINY) d = TINY;
            d = 1.0/d;
            c = 1.0 + aa/c;
            if (Math.abs(c) < TINY) c = TINY;
            delta = d*c;
            h *= delta;

            if (nIter >= BETACF_MAXITER) {
                throw new RuntimeException("betaContFrac: No convergence");
            }
        } while (Math.abs(delta - 1.0) > tol);

        return h;
    }

}
```